

Using an adaptive collection of local evolutionary algorithms for multi-modal problems

Jonathan E. Fieldsend

Received: date / Accepted: date

Abstract Multi-modality can cause serious problems for many optimisers, often resulting convergence to sub-optimal modes. Even when this is not the case, it is often useful to locate and memorise a range of modes in the design space. This is because “optimal” decision parameter combinations may not actually be feasible when moving from a mathematical model emulating the real problem, to engineering an actual solution, making a range of disparate modal solutions of practical use. This paper builds upon our work on the use of a collection of localised search algorithms for niche/mode discovery which we presented at UKCI 2013 when using a collection of surrogate models to guide mode search. Here we present the results of using a collection of exploitative local evolutionary algorithms (EAs) within the same general framework.

The algorithm dynamically adjusts its population size according to the number of regions it encounters that it believes contain a mode, and uses localised EAs to guide the mode exploitation. We find that using a collection of

localised EAs, which have limited communication with each other, produces competitive results with the current state-of-the-art multi-modal optimisation approaches on the CEC 2013 benchmark functions.

Keywords Evolutionary algorithms · multi-modal problems · local search · dynamic populations · self-adaptation.

1 Introduction

Optimisation problems in the real world often exhibit a degree of multi-modality – in a particular volume of design space there may be more than one solution which performs equally as well as another, but the regions *between* these solutions map to quality values distinctly *less* good. These are often conceptualised as *peaks* (or *troughs* if the quality measure is to be minimised) and are often referred to as *niches* or *modes*.

There are a number of reasons why decision makers (DMs) are interested in locating disparate peaks. They can offer insight into the behaviour of the problem they are dealing with, as isolated but equally good design solutions can provide useful information into design parameter interactions. Additionally, by discovering parameter combinations which have

J.E.Fieldsend
Computer Science, University of Exeter, Exeter,
EX4 4QF, UK
Tel.: +44-1392-722090
Fax: +44-1392-217965
E-mail: J.E.Fieldsend@exeter.ac.uk

the equivalent (or similar) behaviour, but which are distributed widely in design space, a certain degree of robustness is provided to the DM. This is because often the model optimised is a software emulation of a physical process, whose mapping may not be exact in all regions. Therefore obtaining a wide spread of good solutions mitigates the risk that realising a particular parameter combination may turn out to be infeasible from a manufacturing point of view — or when manufactured may not behave as emulated.

The paper proceeds as follows. In Section 2 we provide a short description of the general multi-modal optimisation problem, this is followed by a short overview of evolutionary optimisation for multi-modal problems in Section 3. In Section 4 our multi-modal optimisation algorithm is described, and in Section 5 its empirical performance is compared to that of a number of recently developed algorithms on a range of standard test problems. In Section 6 we discuss the sensitivity of the algorithm to its meta-parameters. The paper concludes with a discussion in Section 7.

2 Multi-modal landscapes

Many methods have been developed for locating multiple optima. These range from the basic ‘Multistart’ approach, which applies local search from randomly generated locations (see discussion in Liang et al. (2000)) – to more sophisticated techniques like fitness sharing and crowding, which both fall in the broader area of *niche* methods. Holland (1975) first introduced the fitness sharing concept, which was later refined as a means to partition the search population into different subpopulations based on their fitness (Goldberg and Richardson 1987). A succinct overview of these general ideas is presented in Sareni and Krähenbühl (1998).

The general aim in multi-modal optimisation is similar to that of standard optimisation, that is, given a legal search domain \mathcal{X} , without

loss of generality, we seek to maximise

$$f(\mathbf{x}), \quad \mathbf{x} \in \mathcal{X} \quad (1)$$

subject to

$$\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \dots, g_M(\mathbf{x})) = 0, \quad (2)$$

$$\mathbf{e}(\mathbf{x}) = (e_1(\mathbf{x}), \dots, e_J(\mathbf{x})) \geq 0. \quad (3)$$

In the case of a multi-modal problem, we seek not simply to discover a single \mathbf{x} which maximises $f(\mathbf{x})$ given the constraints (2) and (3), but *all* $\mathbf{x}^* \in \mathcal{X}$ which obtain the maximum possible function response, but which inhabit isolated peak regions. That is, the mapped objective values in the immediate region of an \mathbf{x}^* are all lower than $f(\mathbf{x}^*)$. Local optima (local modes/peaks) are locations which are surrounded in the immediate vicinity with less ‘fit’ solutions (lower responses from $f(\cdot)$), but which do not themselves have the highest possible fitness obtainable in the wider legal domain.

3 Recent Trends in multi-modal optimisation

Many methods exist to search for multiple optima, with fitness-sharing and crowding (Sareni and Krähenbühl 1998) being two popular evolutionary computation (EC) approaches. Essentially these promote regional subpopulations of a search population, which are concerned with optimising separate modes. As recently highlighted in Li and Deb (2010), these algorithms are often highly parameterised, and rely on well-chosen values to perform as required. Other approaches that have been developed for use within evolutionary algorithms (EAs) include clustering (Yin and Germany 1993), de-rating (Beasley et al. 1993), restricted tournament selection (Harik 1995), speciation (Li et al. 2002), and stretching and deflation (Parsopoulos and Vrahatis 2004).

The current state-of-the-art (based upon the results of the CEC 2013 competition in the field) rely on a range of different technologies and heuristics to maintain, search for

and exploit mode estimates. The best performing multimodal optimiser, proposed by Preuss (2010), utilises the covariance matrix adaptation evolution strategy (CMA-ES) of Hansen and Ostermeier (2001). Standard CMA-ES itself was ranked third overall in the CEC 2013 competition. CMA-ES in its basic form learns successful mutation steps by implementing a principle component analysis of the previously selected mutation steps to determine the mutation distribution to use at its next time step. By adapting the mutation distribution as the algorithm proceeds it can rapidly converge on a mode location local to its converging population. To mitigate stagnation on a *local* optima the algorithm can restart in a different region of search space. Rather than electing the restart location at random, Preuss (2010) uses nearest-better clustering to partition a search population into sub groups concerned with different modes. This is facilitated by fitting a spanning tree on the population, linking all population members to their nearest neighbour (in design space) which is better, and disconnecting the longest edges (thus assuming that the best search points on different peaks are likely to be further away from each other than neighbours on the same peak). This leads to another property of this approach – it is *dynamic* in the number of modes it maintains and returns (although limited to a maximum number, set *a priori*). The next best performing multi-modal optimiser is also dynamic in its mode maintenance (Epitropakis et al. 2013), storing an external dynamic archive of estimated mode locations which supports a reinitialisation mechanism, along with an adaptive control parameter technique for the differential evolution algorithm driving the search. Finally, the fourth ranked algorithm proposed by Molina et al. (2013) uses an external memory to store the current global optima, along with an adaptive niche radius to mitigate the effect of setting this parameter *a priori* to a value which may not be appropriate to the problem at hand. A mesh of solutions is exploited, with a combination method that generates solutions

in the direct of nearest (estimated) global optima.

It should be noted that the published ranking of these algorithms is derived from their average performance on the twenty problem formulations used in the CEC 2013 benchmark suite, averaged across five different accuracy levels for fixed numbers of function evaluations. Given different test problems, and/or a different number of permitted function evaluations and accuracy levels, there may be a different ranking obtained. The top ranked algorithms do however possess a number of similar characteristics which would seem to describe an effective multi-modal optimiser, namely: self-adaptation of search parameters, dynamic mode maintenance, and exploitative local search. Here we leverage and develop these ideas, and build on our recent work in the area (Fieldsend 2013), which fitted local surrogate models to regions of the design space to guide a distributed local search. This was assessed on 12 variants of eight multi-modal test problems from the literature, and was found to provide equivalent or better performance to the optimisers described in Clerc and Kennedy (2002); Secrest and Lamont (2003); Li and Deb (2010); Thomsen (2004); Epitropakis et al. (2011a) – based on the published results of these algorithms in Li and Deb (2010) and Li et al. (2013a).

Here we further develop the localised search evolutionary algorithm (LSEA) framework we introduced in (Fieldsend 2013), and present the use of exploitative hill-climbing evolutionary algorithms rather than surrogate models to drive the local search within the framework. We label the algorithm we describe here as LSEA_{EA} to denote its use of EAs as its local search mechanism, distinct from our work which utilised Gaussian Processes (LSEA_{GP}).

The number of local modes to use is not predefined, and is not limited, allowing the algorithm to *learn* the degree of multi-modality as it searches the design space. It accomplishes this by merging and splitting regions covered by the local EAs, and by searching in new regions via both recombination and specula-

tively looking in new areas. The algorithm is highly *self-adaptive*, at both the global and local level, and does not require parameters such as niche radius, number of niches, mutation widths, etc. However it still requires some meta-parameters, its sensitivity to which is discussed in Section 6.

4 Proposed algorithm

Rather than employ a single EA to search the entire cost landscape, here we take the approach of having *many* localised EAs, who are concerned only with the local landscape surrounding each of the currently estimated niches. These EAs operate in parallel, and their number is dynamic as the search progresses, and is not limited to any pre-defined maximum.

Because the EAs are confined to small volumes of search space local to a mode estimate, we only use previously evaluated solutions in their immediate vicinity to learn the parameters employed by the local EAs, thus even though there may be many local EAs proposing new solutions at each iteration, the self-adaptation of the EA parameters for each local optimiser is rapidly accomplished. The overarching algorithm (which maintains the local EAs) is also self-adaptive, with the number of niches maintained depending on the properties of the landscape discovered, and it seeks to balance both exploration of the space for previously undiscovered niches, and exploitation of the niches found thus far. As such, the number of niches in the search space does not need to be known *a priori*, or a vast search population maintained right from the start – instead the number of niches will dynamically shrink or expand as the search progresses. The algorithm maintains a set of sets of *niche histories* \mathbf{X} , where \mathbf{X}_i is the set of m designs $\{\mathbf{x}\}_{j=1}^m$ which define the immediate peak region of the i th niche. These are exploited in the search process by each local EA, and are used to adaptively update their search parameters. Each \mathbf{X}_i therefore includes a single mode esti-

Algorithm 1 High-level pseudocode of the localised search evolutionary algorithm.

Require: $n, max_evals, rep, max_hist$

```

1:  $\mathbf{X} := \text{latin\_hypercube\_samp}(n)$ 
2:  $\mathbf{Y} := \text{evaluate}(\mathbf{X})$ 
3:  $evals := n$ 
4:  $t := 0$ 
5: while  $evals < max\_evals$  do
6:    $\{\mathbf{X}, \mathbf{Y}, evals\} := \text{compare\_niches}(\mathbf{X}, \mathbf{Y}, evals)$ 
7:    $\{\mathbf{X}, \mathbf{Y}\} := \text{trim\_histories}(\mathbf{X}, \mathbf{Y}, max\_hist)$ 
8:    $\mathcal{E} := \text{update\_local\_EAs}(\mathbf{X}, \mathbf{Y})$ 
9:    $\{\mathbf{X}, \mathbf{Y}, evals\} :=$ 
      $\text{exploit\_local\_EAs}(\mathcal{E}, \mathbf{X}, \mathbf{Y}, evals)$ 
10:   $t := t + 1$ 
11:  if  $t = rep$  then
12:     $\{\mathbf{X}, \mathbf{Y}, evals\} :=$ 
       $\text{crossover\_niches}(\mathbf{X}, \mathbf{Y}, evals)$ 
13:     $t := 0$ 
14:  end if
15:   $\{\mathbf{X}, \mathbf{Y}, evals\} :=$ 
     $\text{generate\_random\_niche}(\mathbf{X}, \mathbf{Y}, evals)$ 
16: end while
17:  $\{X^*, Y^*\} := \text{extract\_peak\_members}(\mathbf{X}, \mathbf{Y})$ 
18: return  $X^*, Y^*$ 
```

mate, \mathbf{x}^* . \mathbf{Y}_i is the collection of corresponding responses from $f(\cdot)$ for the elements of \mathbf{X}_i .

The algorithm at a high-level is described in Algorithm 1. The initial number of random solutions to be evaluated, n , is inputted (these form the basis of the initial estimated niches), along with the maximum number of function evaluations permitted for the algorithm. The algorithm proceeds as follows. The initial solutions are drawn from latin hypercube sampling of the search space (line 1), which also form the initial niche histories. Following the initialisation the main optimisation loop is on lines 5-16. On line 6 the niches are compared to one another – and those that are deemed to be concerned with the *same* niche are merged (this is detailed further in Algorithm 2). After niche merging, any local history exceeding the *max_hist* size is trimmed to the maximum – the worst performing local history members being removed. After this, the collection of local EAs, \mathcal{E} , have their parameters updated based on their respective \mathbf{X}_i and \mathbf{Y}_i (line 8, and Algorithm 3), and on line 9 these local EAs are used to proposed a new so-

Algorithm 2 The `compare_niches` subroutine.

Require: $\mathbf{X}, \mathbf{Y}, evals$

```

1:  $\{X^*, Y^*\} := \text{extract\_peak\_members}(\mathbf{X}, \mathbf{Y})$ 
2: index members in  $\mathbf{X}^*$  whose location has moved
   in the last generation
3: find the closest other niche to each changed
   niche
4: for each selected niche pair  $\mathbf{X}_i^*$  and  $\mathbf{X}_j^*$  do
5:   if  $\text{distance}(\mathbf{X}_i^*, \mathbf{X}_j^*) \leq \text{tolerance}$  then
6:     merge niches
7:   else
8:     find the midpoints,  $\mathbf{x}'$ , for the paired
       niches
9:     evaluate midpoint for paired niches,  $y' =$ 
        $f(\mathbf{x}')$ 
10:     $evals := evals + 1$ 
11:    if  $y' < Y_i^* \wedge y' < Y_j^*$  then
12:      add  $\mathbf{x}'$  to either  $\mathbf{X}_i$  or  $\mathbf{X}_j$ 
13:    else
14:      merge niches
15:    end if
16:  end if
17: end for
18: return  $\mathbf{X}, \mathbf{Y}, evals$ 
```

lution for evaluation in the vicinity of each current niche estimate. Each of these is then evaluated on the actual objective function, $f(\cdot)$, and these are used to improve the niche estimate further (details of proposal selection are provided in Algorithm 4).

Every *rep* generations the niche peaks are crossed over (line 12) using simulated binary crossover, SBX, (Deb and Agrawal 1994). The resulting children are then placed in new niches (although these may subsequently be merged when the algorithm loops back to line 6).

The `compare_niches` subroutine, as outlined in Algorithm 2, is concerned with *reducing* the number of niches maintained, in case, for instance, two niches are climbing up the same peak from different directions (and therefore are concerned in actuality with the *same* peak). To this end the method first marks all niches whose peaks have changed since the last generation (for instance, due to the niche being brand new, or a higher point in the niche locality being found by its local EA in the previous iteration). The closest *neighbouring* niche peak (in design space) to each of these

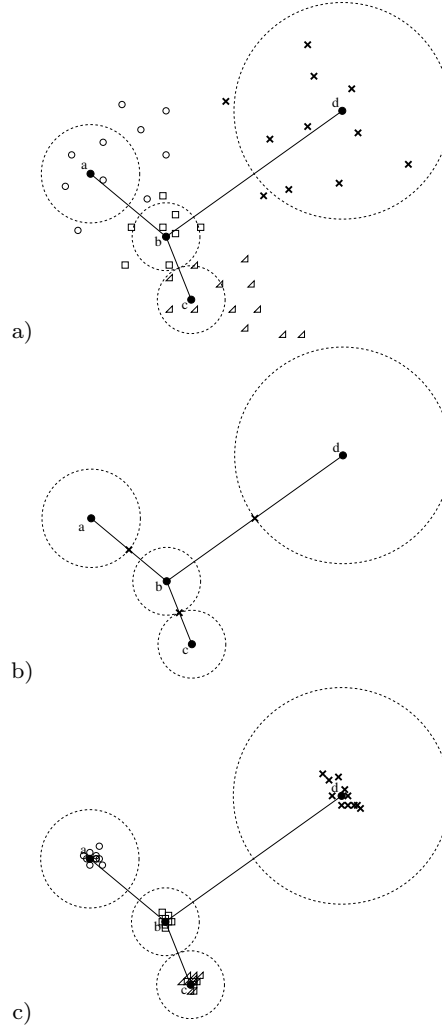


Fig. 1 Illustration of the use of local modes in the optimiser. (a) Four estimated peaks are highlighted in the region of 2D search space that is illustrated. These are marked with filled circles and labelled *a*, *b*, *c* and *d*. Each of these modes possesses a niche history which are displayed with a unique symbol (circles, squares, triangles and crosses). Hyperspheres with radii equal to half the distance to the closest peak neighbour are placed around each peak, and marked with a dashed line. (b) The mid point between each mode and its closest neighbouring mode is calculated. These mid points are evaluated to see if the modes should be merged. (c) Over time the region covered by the history stored for each mode contracts (due to the limit on history size), allowing refinements of the mode estimates. New modes may enter through random proposals and via crossover.

Algorithm 3 The `update_local_EAs` subroutine.

Require: \mathbf{X}, \mathbf{Y}

```

1: for each niche history set  $\mathbf{X}_i$  do
2:   set the  $d$  parameter of  $\mathcal{E}_i$  to Euclidean distance from  $i$ th mode location to nearest next mode location
3:   if  $|\mathbf{X}_i| > 1$  then
4:     set the  $\mathbf{w}$  of  $\mathcal{E}_i$  equal to the variance of  $\mathbf{X}_i$ 
5:   end if
6: end for
7: return  $\mathcal{E}$ 

```

niches is found. Once all pairs have been identified the procedure then processes each pair in turn¹ (lines 4-17). If the two peaks are within some small tolerance of each other in design space (here set at 1.0×10^{-6}), then the niches are automatically merged (lines 5-6).² If paired peak estimates are larger than the tolerance apart the location midway between the two peaks is evaluated (lines 8-10). The midpoint, \mathbf{x}' , is then compared to its two parents. If it is worse than both the niches are maintained separately and the evaluated midpoint is simply added to the niche history of one of the peaks (as it may prove useful to fitting their local EA later). However if it is better than one or both of its parents, then the niches are merged. The mid point is added to the history of the resultant merged niche, and becomes its \mathbf{x}^* estimate if appropriate. Conceptually this is akin to the hill-valley approach introduced by Ursem (1999), however the overhead of its use is considerably lower in the work here (as only a single point is used on the line between two locations, it is only used between niche peak estimates, and then only when their peak estimate locations have shifted).

In the `update_local_EAs` subroutine (Algorithm 3) is concerned with maintaining the parameters of each local EA, \mathcal{E}_i . Each \mathcal{E}_i is concerned with searching a local region of de-

Algorithm 4 The `exploit_local_EAs` subroutine.

Require: $\mathcal{E}, \mathbf{X}, \mathbf{Y}, evals$

```

1:  $u := \mathcal{U}(0, 1)$ 
2: if  $u < \frac{1}{2}$  then
3:   for each local EA  $\mathcal{E}_i \in \mathcal{E}$  do
4:      $\{\mathbf{X}, \mathbf{Y}, evals\} :=$ 
        $local\_EA\_proposal(\mathcal{E}_i, \mathbf{X}, \mathbf{Y}, evals)$ 
5:   end for
6: else
7:    $\mathcal{E}^* = get\_EAs\_for\_best\_niches(\mathcal{E}, \mathbf{Y})$ 
8:    $preds := 0$ 
9:   while  $preds < |\mathcal{E}|$  do
10:    for each local EA  $\mathcal{E}_i^* \in \mathcal{E}^*$  do
11:       $\{\mathbf{X}, \mathbf{Y}, evals\} :=$ 
         $local\_EA\_proposal(\mathcal{E}_i^*, \mathbf{X}, \mathbf{Y}, evals)$ 
12:       $\mathcal{E} := update\_local\_EAs(\mathbf{X}, \mathbf{Y})$ 
13:    end for
14:     $preds := preds + |\mathcal{E}^*|$ 
15:  end while
16: end if
17: return  $\mathbf{X}, \mathbf{Y}, evals$ 

```

sign space, centred on the peak estimate that it is currently contains. For each \mathcal{E}_i the distance from its mode location estimate to the next closest mode location estimate is stored in d . This is used to restrict search to the immediate neighbourhood of the estimate mode (keeping the search local). If there is only a single location in the mode history (i.e. it is a brand new proposed mode region), then d is also used to set the initial mutation width value for each search space dimension. If however there is a larger history stored for that niche, then the mutation width vector \mathbf{w} is set to the standard deviation of the stored history \mathbf{X}_i on each design dimension. Due to the trimming of the worst performing solutions via the *max_hist* threshold used in the main algorithm, the values in the width vector will tend reduce as the search focuses on the peak region.

Following the updating of the local EA parameters, the local EAs are then used to propose a new solution for each mode, drawn in the immediate vicinity of the niche peak. This is described in Algorithm 4. Half of the time, all local niches have a single proposal generated by their local EA (lines 2-5). The other half of the time the algorithm focuses on the

¹ Note, to avoid wasted computation is best to check the list produced, as two changed points may be the closest to each other, thus duplicating pairs.

² The tolerance of 10^{-6} means that the algorithm can effectively preserve up to 10^{6D} peak estimates in any unit cube of D -dimensional design space.

Algorithm 5 The `local_EA_proposal` subroutine.

Require: $\mathcal{E}_i, \mathbf{X}, \mathbf{Y}, evals$

```

1:  $\{\mathbf{x}^*, d, \mathbf{w}\} = \text{extract}(\mathbf{X}_i, \mathcal{E}_i)$ 
2:  $\mathbf{x} := \mathbf{x}^*$ 
3: if  $|\mathbf{X}_i| = 1$  then
4:   repeat
5:      $\mathbf{x} = \mathbf{x}^* + \mathcal{G}(0, 0.1d^2)$ 
6:   until  $\mathbf{x} \in \mathcal{X}$  and  $\text{distance}(\mathbf{x}, \mathbf{x}^*) < \frac{d}{2}$ 
7: else
8:    $u := \mathcal{U}(0, 1)$ 
9:   if  $u < \frac{1}{2}$  then
10:    set  $k$  to be the search space dimension
    where  $\mathbf{w}$  has its largest value
11:    repeat
12:       $\mathbf{x}_k := \mathbf{x}_k^* + \mathcal{G}(0, w_k)$ 
13:    until  $\mathbf{x} \in \mathcal{X}$  and  $\text{distance}(\mathbf{x}, \mathbf{x}^*) < \frac{d}{2}$ 
14:  else
15:    set  $k$  to be a random search space dimension
16:    repeat
17:       $\mathbf{x}_k := \mathbf{x}_k^* + \mathcal{G}(0, \max(\text{distance}(\mathbf{X}_i)^2)$ 
18:    until  $\mathbf{x} \in \mathcal{X}$  and  $\text{distance}(\mathbf{x}, \mathbf{x}^*) < \frac{d}{2}$ 
19:  end if
20: end if
21: evaluate  $\mathbf{x}$  on problem, to obtain  $y$ 
22:  $evals := evals + 1$ 
23:  $\{\mathbf{X}, \mathbf{Y}\} := \text{update\_niche}(\mathbf{X}, \mathbf{Y}, i, \mathbf{x}, y)$ 
24: return  $\mathbf{X}, \mathbf{Y}, evals$ 

```

currently best performing niches. In the original LSEA (Fieldsend 2013) it was noted that as the algorithm sought to find *all* modes, in landscapes where there were many local modes the search could be diluted away from the global peaks. The introduction of elitism into the algorithm attempts to mitigate this. We compare a number of different regimes for elite selection (line 7) in Section 5.1. The elite selection procedure is repeatedly used to propose solutions (and update \mathbf{X} , \mathbf{Y} and the corresponding peak estimates) until the number of function evaluations used reaches the number that would have been exhausted were the entire collection of local EAs to have been exploited (lines 9-15).

The manner in which a local EA proposes a new solution is described in Algorithm 5. Initially the proposal location is set equal to the current mode location for a particular local EA (line 1). If the mode is new, and only has one

solution in its tracked history (i.e. the mode itself), then a proposed location is simply perturbed by adding a draw from an isotropic Gaussian of the same dimensionality as \mathbf{x} , whose variance is a tenth of the distance to the next closest mode location. This is continued until a legal solution is generated, that lies within the hypersphere centred on \mathbf{x}^* (the current mode estimate of the i th niche) whose radius is half the distance to the next closest mode location (i.e. $\frac{d}{2}$). This ensures that proposed solutions are always in the locality of the current mode (although, as the estimated mode location shifts over time, and modes are merged – or new modes appear, this radius can vary). If the mode location is not new, then the mutation widths stored by the local EA based upon the local history are used. A single element of \mathbf{x}^* is altered in the child solution, either by selecting the dimension with the greatest variation in \mathbf{X}_i and perturbing it with a draw with the corresponding variance (line 9-12), or by selecting a decision parameter at random, and perturbing it with a Gaussian draw whose variance is derived from the maximum distance from \mathbf{x}^* to its tracked history members. As the mode history converges, both these variance terms will reduce, prompting a finer resolution on the mode estimate. Additionally, the second drawing method allows search to continue on parameters whose values in \mathbf{X}_i have all collapsed to the same location.³

Before returning, the `local_EA_proposal` subroutine passes each of the proposed solu-

³ As rejection sampling is used to ensure the locality of the proposal to the niche it is evolved from, the number of rejections at a location is tracked. If there is an excessive number of rejections in a row for the child generation method (we use 10 here), the perturbation on lines 3-5 is used instead. This situation can occur if new modes have appeared over time making the history at a particular peak cover a region in the search space which exceeds the niche radius. The situation of switching child generation due to excessive rejections will however resolve itself as better solutions are found around the peak (updating the history and pulling it inside the niche radius), or due to the closest neighbour being *merged*, enlarging the peak radius once more.

Algorithm 6 The `update_niche` subroutine.

Require: $\mathbf{X}, \mathbf{Y}, i, \mathbf{x}, y$
1: $\{\mathbf{x}^*, y^*\} := \text{get_peak}(\mathbf{X}_i, \mathbf{Y}_i)$
2: **if** $y > y^*$ **then**
3: replace peak with \mathbf{x}
4: **end if**
5: update niche history with \mathbf{x} and y
6: **return** \mathbf{X}, \mathbf{Y}

Algorithm 7 The `crossover_niches` subroutine.

Require: $\mathbf{X}, \mathbf{Y}, \text{evals}$
1: **return** $\mathbf{X}, \mathbf{Y}, \text{evals}$
2: $\{X^*, Y^*\} := \text{extract_peak_members}(\mathbf{X}, \mathbf{Y})$
3: $I :=$ random permutation of the indices of the niche sets
4: **while** $|I| > 1$ **do**
5: remove the last two values held in I (i and j)
6: crossover X_i^* and X_j^* to create \mathbf{x}' and \mathbf{x}''
7: evaluate the offspring, \mathbf{x}' and \mathbf{x}''
8: create a new niche for each of the offspring
9: $\text{evals} := \text{evals} + 2$
10: **end while**

tions and their evaluations to the `update_niche` subroutine, which ascertains whether the new solution has improved the peak estimate (described in Algorithm 6). If the new location is worse than the current peak, then the proposal is added to the history of the current niche (although it may be truncated at a later point in the main algorithm, if the history is at capacity and it is not better than any other history member).

The main exploration driver occurs via niche crossover, as described in Algorithm 7 (the SBX parameter was set to 20 in our empirical work), with additional speculative search via a random element in each generation (line 14 of Algorithm 1). The local EAs provide the main exploitation driver. Note, the algorithm does not concern itself with how fit the niches it maintains are in relation to each other, but rather that the niches are *locally* fit with respect to the the immediate vicinity around a niche peak. As such this algorithm will successfully find many peaks of varying heights, although the modification in Algorithm 4 from (Fieldsend 2013) means it will bias the search

every other algorithm iteration toward ‘fitter’ niches.

5 Empirical results

At the end of this section we compare LSEA_{EA} to a range of state-of-the-art evolutionary multimodal optimisers. First however, we examine how incorporating different forms of elitism varies the performance of the optimiser.

5.1 Elitism approaches

Results from our UKCI 2013 work indicated that the algorithm struggled with problems with very many local peaks – as it attempts to optimise them all in parallel. To mitigate against this, we compare the use of four different procedures for the `get_EAs_for_best_niches` function in Algorithm 4. These are as follows:

- the baseline approach which to returns all local EAs (i.e., no elitism);
- selecting the best $p\%$ EAs, defined by *range*;
- selecting the best $p\%$ EAs, defined by *rank*;
- selecting the best p EAs.⁴

Performance is evaluated on the 20 benchmark problems of the CEC 2013 “Competition on Niching Methods for Multimodal Optimization” (Li et al. 2013a,b). The 20 benchmark problems are of varying dimensionality and number of optima, and are derived from 12 base test problems, as defined in Table 1, and illustrated in Figure 2 (via 1D and 2D mappings). As can be seen, the Equal Maxima, Himmelblau, Vincent and Modified Rastrigin problems *only* have global peaks. The Five-Uneven-Peak Trap, Uneven Decreasing Maxima, Shubert, and Composite Functions 1-4 (which combine properties of a number of different problems, in different regions of design space) all have local maxima as well as global maxima (with the later five functions having

⁴ If p exceeds the total number of local EAs at a generation, all local EAs are returned.

Table 1 Test functions. Properties as described in Li et al. (2013a).

| Name | function # | Dim. | Glob. Opt. | Properties |
|--------------------------|--------------------|--------------|------------|--|
| Five-Uneven-Peak Trap | F1 | 1 | 2 | Simple, deceptive |
| Equal Maxima | F2 | 1 | 5 | Simple |
| Uneven Decreasing Maxima | F3 | 1 | 1 | Simple |
| Himmelblau | F4 | 2 | 4 | Simple, non-scalable, non-symmetric |
| Six-Hump Camel Back | F5 | 2 | 2 | Simple, non-scalable, non-symmetric |
| Shubert | F6, F8 | 2, 3 | 18, 81 | Scalable, number of optima increase with search dimension, unevenly distributed grouped optima |
| Vincent | F7, F9 | 2, 3 | 36, 216 | Scalable, number of optima increase with search dimension, unevenly distributed optima |
| Modified Rastrigin | F10 | 2 | 12 | Scalable, number of optima independent from search dimension, symmetric |
| Composition Function 1 | F11 | 2 | 6 | Scalable, separable, non-symmetric |
| Composition Function 2 | F12 | 2 | 8 | Scalable, separable, non-symmetric |
| Composition Function 3 | F13, F14, F16, F18 | 2, 3, 5, 10 | 6 | Scalable, non-separable, non-symmetric |
| Composition Function 4 | F15, F17, F19, F20 | 3, 5, 10, 20 | 8 | Scalable, non-separable, non-symmetric |

Table 2 Parameters used for performance measurement, and maximum number of function evaluations per optimiser run.

| Function | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 |
|-------------|------|------|------|------|---------|---------|------|-----------|------|------|
| r | 0.01 | 0.01 | 0.01 | 0.01 | 0.5 | 0.5 | 0.2 | 0.5 | 0.2 | 0.01 |
| Peak height | 200 | 1 | 1 | 200 | 1.03163 | 186.731 | 1 | 2709.0935 | 1 | -2 |
| Max evals | 50k | 50k | 50k | 50k | 50k | 200k | 200k | 400k | 400k | 200k |
| Function | F11 | F12 | F13 | F14 | F15 | F16 | F17 | F18 | F19 | F20 |
| r | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| Peak height | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Max evals | 200k | 200k | 200k | 400k | 400k | 400k | 400k | 400k | 400k | 400k |

many more local maxima than global maxima). Due to space constraints formal definitions are not provided here – however a technical report detailing them (Li et al. 2013a), can be found online.

We follow the algorithm assessment protocol used in the CEC 2013 competition. Problem assessment criteria are detailed in Table 2. The parameter r gives the maximum distance (in design space) a solution may be from a

peak be categorised to have found it – subject to a further *accuracy* level, ϵ , which gives the maximum distance from the global maximum in objective space. For all problems five different accuracy levels are assessed, $\epsilon = \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$.

As in the original competition, we run our algorithm 50 times on each problem. Here we use the *peak ratio* (PR) measure to assess the performance of the different variants. The PR

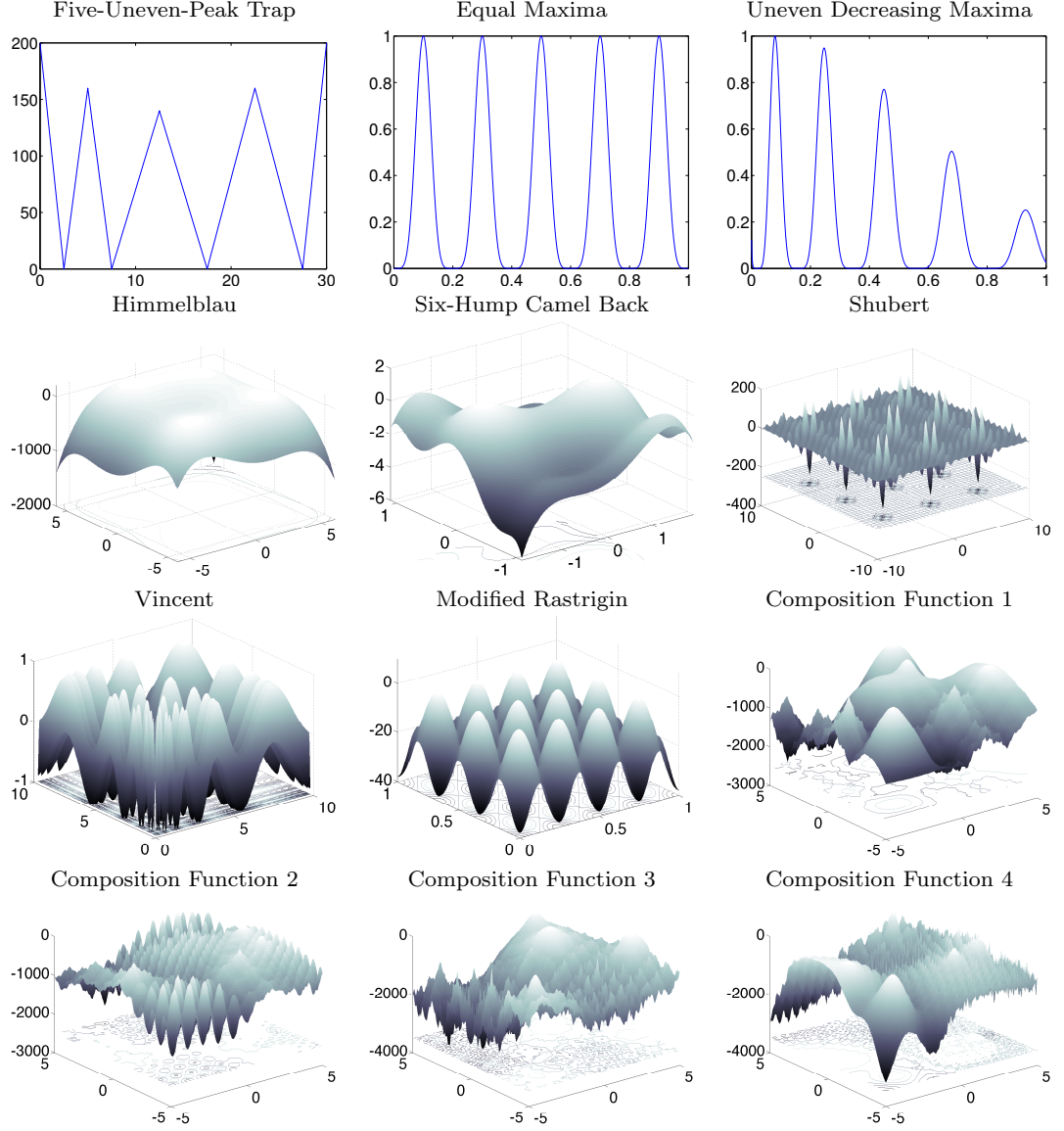


Fig. 2 Test function landscapes. Darker greys indicate low fitness and lighter greys indicate high fitness.

is the average proportion of global peaks found across runs, i.e. for q runs:

$$PR = \frac{\sum_{i=1}^q g_i}{tq} \quad (4)$$

where g_i denotes the number of global optima discovered by the i th run, and t is the total number of global peaks. We use the code made available by the CEC 2013 competition organ-

isers for representing the test problems, and for assessing algorithm performance.⁵

Table 3 shows the average, median and standard deviation of the PR measure for $LSEA_{EA}$ when using different elitism functions (averaged over the mean performance on each of the

⁵ Obtainable from <http://goanna.cs.rmit.edu.au/~xiaodong/cec13-niching/competition/>.

Table 3 Average results across all test problems and accuracy levels of LSEA_{EA} on the peak ratio measure, incorporating different elitism procedures.

| Algorithm | Median | Mean | St. D. |
|--|---------------|---------------|---------------|
| LSEA _{EA} (no elitism) | 0.8108 | 0.6358 | 0.4165 |
| LSEA _{EA} (with search biased to top 1% range) | 0.9536 | 0.7139 | 0.3564 |
| LSEA _{EA} (with search biased to top 10% range) | <u>0.9850</u> | 0.7641 | 0.3086 |
| LSEA _{EA} (with search biased to top 1% rank) | 0.8304 | 0.7460 | 0.2972 |
| LSEA _{EA} (with search biased to top 10% rank) | 0.9433 | 0.7714 | 0.2876 |
| LSEA _{EA} (with search biased to top 10 peaks) | 0.9442 | 0.7516 | 0.3030 |
| LSEA _{EA} (with search biased to top 100 peaks) | 0.8977 | <u>0.7786</u> | <u>0.2781</u> |

20 test problems). As can be seen *all* elitism procedures improve the algorithm performance from the baseline of no elitism – however there is a degree of variation amongst the different approaches.

Figure 3 helps us to examine the behaviour of the different approaches more clearly, and gauge the trade-offs involved. The variant without elitism can be seen to struggle most with F8, and F15-F20 (which correspond to the problems with very many local optima which can dilute its search for global optima). All the elitism approaches can be seen to improve performance on these — pushing the performance lines upwards (geometrically, the means listed in Table 3 correspond to the average area under the five accuracy level lines in the panels in Figure 3). Note: no variant is seen to attain any peaks at the finest accuracy level on F6.

If the elitism is too concentrated (i.e. best 1%, top 10), it tends to perform relatively less well on problems with many global and local optima (e.g. F6, F8) compared to less concentrated elitism. This because too tight a concentration shifts the algorithm behaviour from trying to optimise all peaks, to (mainly) focusing on too few. Interestingly, the ‘range’ elitism approach has better median performance, but the rank and absolute number approaches have better mean performance. Despite of the better median performance of the range elitism approach, we would prefer the rank and absolute approaches, as they are not problem scale dependent. Of these two, the absolute number has marginally better mean performance, and we now compare this elitist variant (with

search biased toward the best 100 peaks found) to the state-of-the-art results in the literature.

5.2 Comparison to other work

We now compare LSEA_{EA} to the published results of a wide range of multi-modal optimisation algorithms (Deb and Saha 2012; Hansen and Ostermeier 2001; Thomsen 2004; Epitropakis et al. 2013; Ronkkonen 2009; Epitropakis et al. 2011b; Auger and Hansen 2005; Preuss 2010; Molina et al. 2013; Bandaru and Deb 2013), which were also applied to the benchmark problems. Our results are directly compared to the combined competition results, published in Li et al. (2013b). Additionally, LSEA_{EA} is compared to the problem level results which are available in Epitropakis et al. (2013) and Molina et al. (2013) for the niching variable mesh optimisation (N-VMO) and dynamic archiving niching differential evolution algorithm (dADE) respectively.

We additionally use the *success rate* (SR) measure in this section. SR measures the proportion of successful runs (those which find *all* global optima given the prescribed ϵ and r). A value of 1.0 therefore indicates that all 50 runs found all global peaks, whereas a value of 0.5 would indicate that half the runs (25) found all global peaks. In terms of the quality measure employed earlier in Section 5.1, an SR of 1.0 also means a PR of 1.0. However, the PR level cannot be deduced from an SR lower than 1.0 (as a SR of 0.0 may mean every run obtained

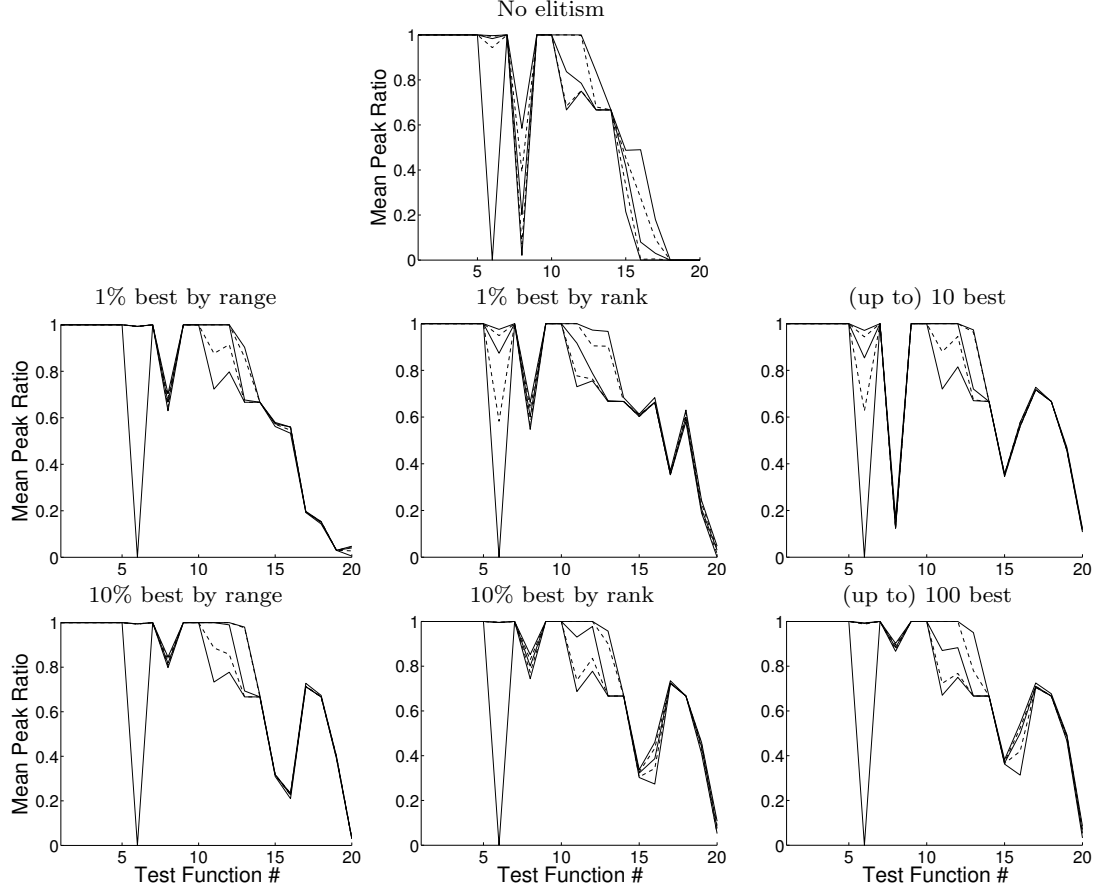


Fig. 3 Performance of $LSEA_{EA}$ with different elitism approaches. Each plot shows the mean PR (over 50 runs) at each of the five accuracy levels, drawn alternately with solid and dashed lines. A tighter accuracy level will always have a lower or equal PR to a wider accuracy level – the lowest line therefore corresponds to $\epsilon = 10^{-5}$, and the highest line to $\epsilon = 10^{-1}$. For many problems these lines are very tightly bunched (or indeed on top of each other); for others (e.g., F11, F12 and F13) they tend to be more spread out.

the majority, but not all, of the peaks, or may have found no peaks whatsoever).

Tables 4 and 5 give the SR and PR results for $LSEA_{EA}$, N-VMO (results from Molina et al. (2013)) and dADE (the nrand/1/bin variant, for which results are provided in Epitropakis et al. (2013)).⁶ A number of interesting properties can be observed from these tables. $LSEA_{EA}$ is seen to perform competitively with the other

two algorithms – it has the best or joint best SR across problems and accuracy levels 49% of the time, compared to 44% for N-VMO and 36% for dADE. For 38% of the problem and accuracy combinations *none* of the algorithms were successful at finding all the peaks on any of their runs (these tended to be the higher accuracy levels on problems with many local modes). $LSEA_{EA}$ has the best or joint best PR results 81% of the time (compared to 50% for N-VMO and 46% for dADE). From the Table 5 it can be seen that for every problem, at each accuracy level, at least one peak was found on

⁶ There appear to be some data entry issues in the tabulated results for the peak ratios of F15 in Epitropakis et al. (2013), as the values reported *increase* from $\epsilon = 10^{-3}$ to 10^{-4} , likewise for F20 from $\epsilon = 10^{-2}$ to 10^{-4} .

Table 4 Success rates of the LSEA_{EA}, N-VMO and dADE/nrand/1/bin algorithms. Best values for each problem and accuracy level among the three algorithms are underlined.

| | F1 | | | F2 | | | F3 | | | F4 | | |
|------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| ϵ | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE |
| 10^{-1} | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> |
| 10^{-2} | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> |
| 10^{-3} | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> |
| 10^{-4} | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> |
| 10^{-5} | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> |
| hline | F5 | | | F6 | | | F7 | | | F8 | | |
| ϵ | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE |
| 10^{-1} | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | 0.880 | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | 0.000 | 0.000 | <u>0.020</u> |
| 10^{-2} | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | 0.8600 | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | 0.240 | 0.000 | 0.000 | 0.000 |
| 10^{-3} | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | 0.840 | 0.360 | <u>1.000</u> | <u>1.000</u> | 0.140 | 0.020 | 0.000 | 0.000 | 0.000 |
| 10^{-4} | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>0.800</u> | 0.000 | 0.780 | <u>1.000</u> | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 10^{-5} | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | 0.000 | 0.000 | 0.000 | <u>1.000</u> | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| | F9 | | | F10 | | | F11 | | | F12 | | |
| ϵ | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE |
| 10^{-1} | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | 0.500 | <u>1.000</u> | <u>1.000</u> | 0.640 | <u>1.000</u> | 0.220 | 0.980 |
| 10^{-2} | <u>1.000</u> | 0.000 | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | 0.380 | <u>1.000</u> | 0.000 | 0.000 | <u>1.000</u> | 0.000 | 0.440 |
| 10^{-3} | <u>1.000</u> | 0.000 | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | 0.280 | <u>0.360</u> | 0.000 | 0.000 | <u>0.260</u> | 0.000 | 0.000 |
| 10^{-4} | <u>1.000</u> | 0.000 | <u>1.000</u> | <u>1.000</u> | <u>1.000</u> | 0.140 | <u>0.040</u> | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 10^{-5} | <u>1.000</u> | 0.000 | <u>1.000</u> | <u>1.000</u> | 0.660 | 0.020 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| | F13 | | | F14 | | | F15 | | | F16 | | |
| ϵ | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE |
| 10^{-1} | 0.700 | <u>1.000</u> | 0.140 | 0.000 | <u>1.000</u> | 0.700 | 0.000 | <u>1.000</u> | <u>1.000</u> | 0.000 | <u>1.000</u> | 0.540 |
| 10^{-2} | <u>0.120</u> | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | <u>0.020</u> | 0.000 | 0.000 | <u>0.020</u> | 0.000 |
| 10^{-3} | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 10^{-4} | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 10^{-5} | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| | F17 | | | F18 | | | F19 | | | F20 | | |
| ϵ | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE |
| 10^{-1} | 0.000 | <u>1.000</u> | 0.760 | 0.000 | <u>0.960</u> | 0.080 | 0.000 | <u>0.220</u> | 0.000 | 0.000 | 0.000 | 0.000 |
| 10^{-2} | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 10^{-3} | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 10^{-4} | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 10^{-5} | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

at least one run by one of the three algorithms – apart from the 2D Shubert function (F6), where not a single run of any algorithm found a peak at the $\epsilon = 10^{-5}$ accuracy level.

A pattern can be seen to emerge for the behaviour of LSEA_{EA} on the PR measure for the more difficult problems with many local modes. At the lowest accuracy level the other algorithms tend to outperform LSEA_{EA}, but as the accuracy required increases their PR performance rapidly drops off, whereas LSEA_{EA} tends to be impacted to a much lesser extent.

Table 6 presents the overall performance assessment of LSEA_{EA} across the 50 runs, 20 test problems and five accuracy levels, and compares these to the results of the 15 state-of-the-art entrants to the CEC 2013 competition (results from (Li et al. 2013b)). When averaged across the test problems LSEA_{EA} is competitive with the current state-of-the-art, the mean PR ranked second overall, and its median PR ranked first.

6 Dynamics and sensitivity of LSEA

We now examine the population dynamics of LSEA, and its sensitivity to its meta-parameters.

6.1 Population dynamics

The effect of the dynamic niche maintenance used in LSEA can be seen when looking at the number of niches maintained by the algorithm. Figure 4 shows the number of estimated niches maintained for the first 500 generations of an example run of LSEA_{EA} for each of the test problems. The number of ‘true’ global optima for each problem is plotted as a horizontal dotted line. The spike of new niches generated via crossover every 10 iterations is clear to see in the panels with the population doubling, and then retracting as a number are merged in subsequent generations.

For those problems that do not possess local optima (F2, F4, F7, F9 and F10), the population can be seen shrink or grow in the ear-

Table 5 Mean peak ratios of the LSEA_{EA}, N-VMO and dADE/nrand/1/bin algorithms. Best values for each problem and accuracy level amongst the three algorithms are underlined.

| ϵ | F1 | | | F2 | | | F3 | | | F4 | | |
|------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 10^{-1} | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 10^{-2} | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 10^{-3} | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 10^{-4} | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 10^{-5} | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| hline | F5 | | | F6 | | | F7 | | | F8 | | |
| ϵ | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE |
| 10^{-1} | 1.000 | 1.000 | 1.000 | 0.993 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.902 | 0.412 | 0.837 |
| 10^{-2} | 1.000 | 1.000 | 1.000 | 0.992 | 1.000 | 1.000 | 1.000 | 1.000 | 0.962 | 0.893 | 0.294 | 0.595 |
| 10^{-3} | 1.000 | 1.000 | 1.000 | 0.991 | 0.940 | 1.000 | 1.000 | 0.945 | 0.892 | 0.884 | 0.270 | 0.545 |
| 10^{-4} | 1.000 | 1.000 | 1.000 | 0.988 | 0.670 | 0.984 | 1.000 | 0.901 | 0.823 | 0.875 | 0.198 | 0.431 |
| 10^{-5} | 1.000 | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.806 | 0.732 | 0.867 | 0.650 | 0.356 |
| | F9 | | | F10 | | | F11 | | | F12 | | |
| ϵ | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE |
| 10^{-1} | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.985 | 1.000 | 1.000 | 0.893 | 1.000 | 0.848 | 0.998 |
| 10^{-2} | 1.000 | 0.683 | 1.000 | 1.000 | 1.000 | 0.978 | 1.000 | 0.667 | 0.667 | 1.000 | 0.745 | 0.887 |
| 10^{-3} | 1.000 | 0.399 | 1.000 | 1.000 | 1.000 | 0.981 | 0.870 | 0.667 | 0.667 | 0.883 | 0.725 | 0.745 |
| 10^{-4} | 1.000 | 0.275 | 1.000 | 1.000 | 1.000 | 0.967 | 0.723 | 0.667 | 0.667 | 0.768 | 0.713 | 0.740 |
| 10^{-5} | 1.000 | 0.192 | 1.000 | 1.000 | 0.968 | 0.947 | 0.670 | 0.667 | 0.667 | 0.750 | 0.565 | 0.728 |
| | F13 | | | F14 | | | F15 | | | F16 | | |
| ϵ | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE |
| 10^{-1} | 0.950 | 1.000 | 0.743 | 0.667 | 1.000 | 0.923 | 0.383 | 1.000 | 1.000 | 0.543 | 1.000 | 0.873 |
| 10^{-2} | 0.780 | 0.667 | 0.667 | 0.667 | 0.667 | 0.667 | 0.373 | 0.713 | 0.620 | 0.523 | 0.703 | 0.667 |
| 10^{-3} | 0.667 | 0.667 | 0.667 | 0.667 | 0.667 | 0.667 | 0.368 | 0.668 | 0.615 | 0.500 | 0.653 | 0.667 |
| 10^{-4} | 0.667 | 0.667 | 0.667 | 0.667 | 0.667 | 0.667 | 0.365 | 0.623 | 0.627 | 0.420 | 0.653 | 0.667 |
| 10^{-5} | 0.667 | 0.663 | 0.667 | 0.667 | 0.637 | 0.667 | 0.363 | 0.390 | 0.620 | 0.313 | 0.633 | 0.667 |
| | F17 | | | F18 | | | F19 | | | F20 | | |
| ϵ | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE | LSEA | N-VMO | dADE |
| 10^{-1} | 0.725 | 1.000 | 0.938 | 0.677 | 0.987 | 0.683 | 0.493 | 0.340 | 0.420 | 0.085 | 0.000 | 0.030 |
| 10^{-2} | 0.710 | 0.475 | 0.472 | 0.667 | 0.483 | 0.660 | 0.490 | 0.133 | 0.143 | 0.070 | 0.000 | 0.000 |
| 10^{-3} | 0.710 | 0.440 | 0.417 | 0.667 | 0.470 | 0.630 | 0.487 | 0.133 | 0.063 | 0.055 | 0.000 | 0.002 |
| 10^{-4} | 0.705 | 0.413 | 0.403 | 0.667 | 0.470 | 0.633 | 0.473 | 0.130 | 0.018 | 0.053 | 0.000 | 0.005 |
| 10^{-5} | 0.705 | 0.320 | 0.410 | 0.667 | 0.360 | 0.627 | 0.467 | 0.103 | 0.000 | 0.033 | 0.000 | 0.000 |

Table 6 Average results across all test problems and accuracy levels of LSEA_{EA}, along with results of state-of-the-art multi-modal algorithms compared in the CEC 2013 competition (detailed in Li et al. (2013b)) on the peak ratio.

| Algorithm | Median | Mean | St. D. |
|--|---------------|---------------|--------|
| LSEA _{EA} (elitist, top 100) | <u>0.8977</u> | 0.7786 | 0.2781 |
| A-NSGA-II (Deb and Saha 2012) | 0.0740 | 0.3275 | 0.4044 |
| CMA-ES (Hansen and Ostermeier 2001) | 0.7750 | 0.7137 | 0.2807 |
| CrowdingDE (Thomsen 2004) | 0.6667 | 0.5731 | 0.3612 |
| dADE/nrand/1 (Epitropakis et al. 2013) | 0.7488 | 0.7383 | 0.3010 |
| dADE/nrand/2 (Epitropakis et al. 2013) | 0.7150 | 0.6931 | 0.3174 |
| DECG (Ronkkonen 2009) | 0.6567 | 0.5516 | 0.3992 |
| DELG (Ronkkonen 2009) | 0.6667 | 0.5706 | 0.3925 |
| DELS-aj (Ronkkonen 2009) | 0.6667 | 0.5760 | 0.3857 |
| DE/nrand/1 (Epitropakis et al. 2011b) | 0.6386 | 0.5809 | 0.3338 |
| DE/nrand/2 (Epitropakis et al. 2011b) | 0.6667 | 0.6082 | 0.3130 |
| IPOP-CMA-ES (Auger and Hansen 2005) | 0.2600 | 0.3625 | 0.3117 |
| NEA1 (Preuss 2010) | 0.6496 | 0.6117 | 0.3280 |
| NEA2 (Preuss 2010) | 0.8513 | <u>0.7940</u> | 0.2332 |
| N-VMO (Molina et al. 2013) | 0.7140 | 0.6983 | 0.3307 |
| PNA-NSGA-II (Bandaru and Deb 2013) | 0.6660 | 0.6141 | 0.3421 |

lier generations to converge to the true number of optima, with the crossover every 10 generations causing the values to diverge, before rapidly returning to the true number of optima

through merging. For problems where there are only a handful of local optima, the population size can be seen to oscillate around the total number of global plus local peaks. For

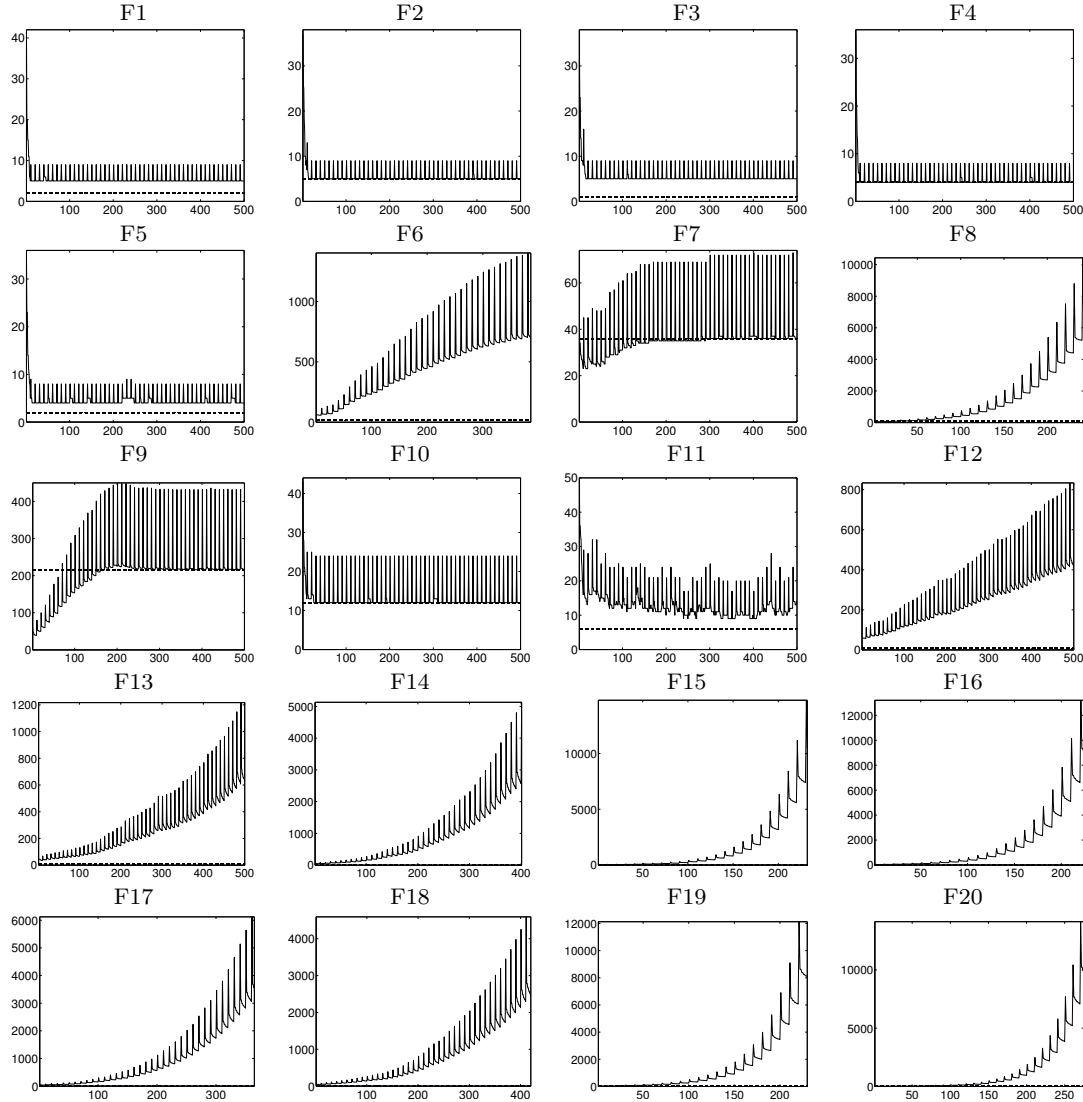


Fig. 4 Maintained niche size per generation for each run (recorded after niche merging). Dashed line shows the total number of global optima for the problem. Population shown for a single run on each problem, for the first 500 algorithm generations (or fewer, if the maximum number of function evaluations was reached).

problems with very many local optima however the population can be seen to be progressively expanding. The best solution bias every other generation means that the global best solutions can be honed proportionally more, but it undoubtedly impacts upon the algorithm searching such a large number of modes in parallel (as can be seen, for a number of the composite problems, the algorithm exhausts

its permitted function evaluations in fewer than 500 generations). As such, although the algorithm is seen to perform well against existing state-of-the-art optimisers, by examining *how* the niche population is evolving there does appear to be scope for further improvement. Initial avenues of further investigation are:

- Using more sophisticated analysis when merging niches or keeping them separate. Currently a basic check is undertaken based on the fitness of a midpoint between two locations, however a surrogate model could also be employed here (although it may require storing the less fit solutions at the edge of a niche, as well as those near the current local maxima).
- Integrate methods to smooth away local optima to mitigate their negative impact on the search (as in for instance Yang and Flockton (1995)).
- Incorporate methods to discard poorly performing niches, and/or regions in \mathcal{X} .

Also there is a potential to develop a heuristic for convergence checking on a particular niche (i.e., exploring when to stop evolving a niche if it is thought the best value in that region has now been attained).

6.2 Sensitivity to meta-parameters

When LSEA has the proposed local EAs embedded (as opposed to surrogates), there are relatively few meta-parameters that need to be set, as both the over-arching algorithm and the local EAs are self-adaptive. However there are a few values that do need setting, which we now examine LSEA’s sensitivity to.

The algorithm is highly insensitive to the number of initial random solutions, n , as it also samples a new random location for a mode proposal at each generation. LSEA_{EA} (and the Gaussian Process variant LSEA_{GP} in Fieldsend (2013)) has been run with $n = 1$ and the algorithm produces results of the same quality to $n = 100$. The main benefit of having $n > 1$ is that at the very earliest part of the run an estimate of the number of modes can be gauged if it is significantly less than n , as the number of modes maintained will decrease rapidly from the start rather than increasing (if $n = 1$ this information does not become apparent until later).

The *rep* value, the number of generations between each crossover expansion of the modes, does have an effect. The left panels in Figure 5 show how the performance of the algorithm varies for different values of *rep* on test functions F6 and F11. On F6 very frequent crossover is beneficial (only at the extreme of crossover every generation does this trend break – as it cannot merge out spurious modes quickly enough). Frequent crossover is beneficial here because F6 is symmetric, and therefore as good peak estimates are located crossover propagates this information rapidly to improve other modes (and to locate undiscovered ones). In contrast on F11, which does not have global peak symmetries the reverse is seen, with more infrequent crossover giving better results (as crossover tends to waste function evaluations in this landscape, by returning poor performing solutions). As landscape properties are unlikely to be known *a priori*, we suggest a *rep* value in the range [10, 50] is used.

The final input is the maximum number of individuals to store in the history of each mode (*max_hist*). The right column of panels in Figure 5 shows the performance of LSEA_{EA} for different values of *max_hist*. The larger the value, the slower the adapted \mathbf{w} values will take to change. This does not seem to be an issue for F6, but does impact the performance on F11. On the other hand, too small a *max_hist* will result in the mutation width varying wildly between generations for a particular mode (affecting convergence), this can be seen to detrimentally affect the PR on both F6 and F11 when *max_hist* < 5. Based on the varying trends seen in the right panels of Figure 5, we would recommend the maximum history for the local EAs to be set between 5 and 20 solutions.

7 Discussion

An new approach to multi-modal optimisation based on using an EA which employs *many* localised self-adaptive EAs has been proposed,

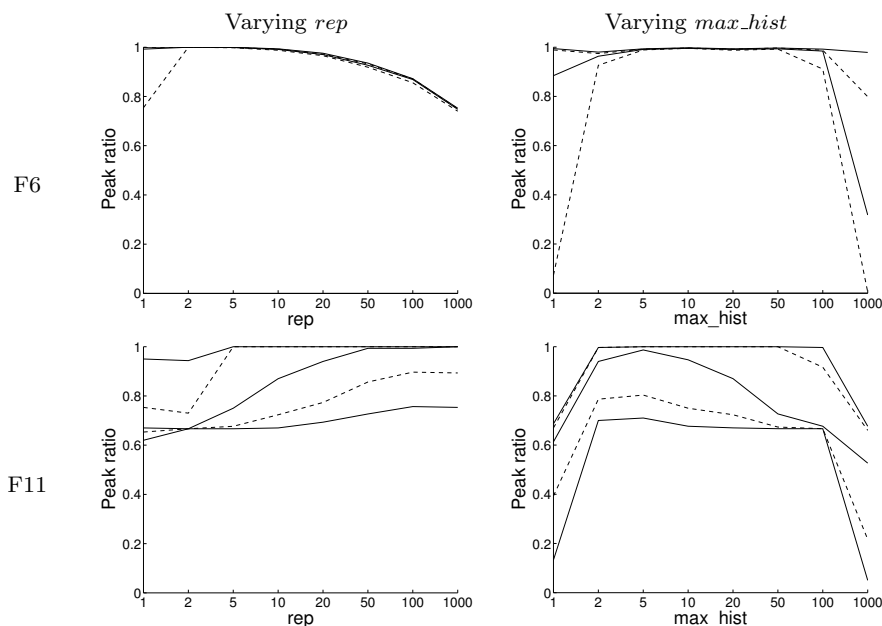


Fig. 5 The effect of different values of rep (left) and different values of max_hist (right) on the performance of $LSEA_{EA}$ on problem F6 (top row) and F11 (bottom row). Results show the mean of 50 runs at $\epsilon = \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. All other algorithm parameters fixed as used in Section 5.2.

and evaluated on a range of problems. One way of viewing it is as a sophisticated distributed hill-climber, albeit one employing a degree of communication between immediately adjacent hills. We have previously used a surrogate to guide the hill traversal, and here use localised EAs, but any search algorithm may essentially be employed to perform the distributed local search within the algorithm. The over-arching EA component is principally concerned with searching for *new* hills to climb, although it will also exploit any symmetry in the landscape. It is however seen to perform well on problems which are non-symmetric too.

Analysis of its behaviour indicates there are areas that can still be improved, with efficiency gains still to be had by improving its niche maintenance subroutines, and routes to tackle potential problem with many local optima have been identified, nevertheless, its performance is competitive with the current state-of-the-art.

MATLAB code for $LSEA_{EA}$ may be found online at <https://github.com/fieldsend>.

8 Acknowledgement

The author would like to express his thanks to Dr. Xiaodong Li, Professor Andries Engelbrecht and Dr. Michael Epitropakis for making the CEC 2013 multi-modal test problems and assessment code directly accessible online (the MATLAB versions of which were used here).

References

- A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *IEEE Congress on Evolutionary Computation*, volume 2, pages 1769–1776, 2005.
- S. Bandaru and K. Deb. A parameterless-niching-assisted bi-objective approach to multimodal optimization. In *IEEE Congress on Evolutionary Computation*, pages 95–102, 2013.
- D. Beasley, D. R. Bull, and R. R. Martin. A sequential niche technique for multimodal func-

- tion optimization. *Evolutionary Computation*, 1(2):101–125, 1993.
- M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multi-dimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6:58–73, 2002.
- K. Deb and R. B. Agrawal. Simulated Binary Crossover for Continuous Search Space. Technical report, Department of Mechanical Engineering, Indian Institute of Technology, Kanpur, IITK/ME/SMD-94027, 1994.
- K. Deb and A. Saha. Multimodal Optimization Using a Bi-objective Evolutionary Algorithm. *Evolutionary Computation*, 20(1):27–62, 2012.
- M. Epitropakis, V. Plagianakos, and M. Vrahatis. Finding multiple global optima exploiting differential evolution’s niching capability. In *IEEE Symposium on Differential Evolution (IEEE Symposium Series on Computational Intelligence)*, pages 80–87, 2011a.
- M. G. Epitropakis, V. P. Plagianakos, and M. N. Vrahatis. Finding multiple global optima exploiting differential evolution’s niching capability. In *IEEE Symposium on Differential Evolution (SDE)*, pages 1–8, 2011b.
- M. G. Epitropakis, X. Li, and E. K. Burke. A dynamic archive niching differential evolution algorithm for multimodal optimization. In *IEEE Congress on Evolutionary Computation*, pages 79–86, 2013.
- J. E. Fieldsend. Multi-Modal Optimisation using a Localised Surrogates Assisted Evolutionary Algorithm. In *UK Workshop on Computational Intelligence (UKCI 2013)*, pages 88–95, 2013.
- D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimisation. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 41–49, 1987.
- N. Hansen and A. Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001.
- G. R. Harik. Finding multimodal solutions using restricted tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 24–31, 1995.
- J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- J.-P. Li, M.E. Balazs, G. T. Parks, and P. J. Clarkson. A species conserving genetic algorithm for multimodal function optimisation. *Evolutionary Computation*, 10(2):207–234, 2002.
- X. Li and K. Deb. Comparing lbest PSO niching algorithms using different position update rules. In *IEEE Congress on Evolutionary Computation*, pages 1564–1571, 2010.
- X. Li, A. Engelbrecht, and M. G. Epitropakis. Benchmark Functions for CEC’2013 Special Session and Competition on Niching Methods for Multimodal Function Optimization. Technical report, Evolutionary Computation and Machine Learning Group, RMIT University, 2013a.
- X. Li, A. Engelbrecht, and M.G. Epitropakis. Results of the 2013 IEEE CEC Competition on Niching Methods for Multimodal Optimization. Presented at 2013 IEEE Congress on Evolutionary Computation Competition on: Niching Methods for Multimodal Optimization, 2013b.
- K.-H. Liang, X. Yao, and C. Newton. Evolutionary Search of Approximated N-Dimensional Landscapes. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 4(3):172–183, 2000.
- D. Molina, A. Puris, R. Bello, and F. Herrera. Variable mesh optimization for the 2013 CEC Special Session Niching Methods for Multimodal Optimization. In *IEEE Congress on Evolutionary Computation*, pages 87–94, 2013.
- K. E. Parsopoulos and M. N. Vrahatis. On the computation of all global minimizers through particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):211–224, 2004.
- M. Preuss. Niching the CMA-ES via Nearest-better Clustering. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO ’10*, pages 1711–1718, 2010.
- J. Ronkkonen. *Continuous Multimodal Global Optimization with Differential Evolution-Based Methods*. PhD thesis, Lappeenranta University of Technology, Finland, 2009.
- B. Sareni and L. Krähenbühl. Fitness Sharing and Niching Methods Revisited. *IEEE Transactions on Evolutionary Computation*, 2(3):97–106, 1998.
- B. Secrest and G. Lamont. Visualizing particle swarm optimization - gaussian particle swarm optimization. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pages 198–204, 2003.
- R. Thomsen. Multimodal optimization using crowding-based differential evolution. In *IEEE Congress on Evolutionary Computation*, pages 1382–1389, 2004.
- R. K. Ursem. Multinational evolutionary algorithms. In *Proceedings of the Congress on Evolutionary Computation*, pages 1633–1640, 1999.
- D. Yang and S. J. Flockton. Evolutionary Algorithms with a Coarse-to-Fine Function Smoothing. In *IEEE International Conference on Evolutionary Computation*, pages 657–662, 1995.
- X. Yin and N. Germany. A fast genetic algorithm with sharing scheme using cluster analysis methods in multi-modal function optimization. In *International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 450–457, 1993.